Tiger Programmable Logic Card (TGPLC)

ASI's Programmable Logic Card (PLC) is a field-programmable card for digital logic, sort of a mini-FPGA inside the Tiger controller. It acts as a single axis device (by default, axis E) and occupies one Tiger address (e.g. 36 in a diSPIM TG-16). Its original use is to enable multi-laser control for diSPIM, but it is designed to be very flexible and can be used for other tasks involving digital signals.

Overview

The PLC has 8 accessible I/Os in the form of BNC connectors on the front panel, as well as connections to the Tiger backplane TTL lines (used in diSPIM for camera triggers and inter-card communication). The PLC also has an array of 16 programmable logic cells[1)]; each logic cell has up to 4 inputs [2)]. Inputs to each cell are selectable from the outputs of any logic cell, BNC inputs, and the Tiger backplane. Cell outputs can be routed to the BNC connectors or to the Tiger backplane. The core of each cell is a user-programmable logic element; possibilities include combinatorial logic, flip flops, programmable one-shot and delay elements. There are 15 different types of cells available.

The PLC periodically runs an evaluation cycle. During each evaluation cycle, the card does the following:

1. Updates external I/Os with the outputs from the prior evaluation cycle [3)]
2. Samples all inputs synchronously [4)]
3. Computes the new outputs one cell at a time, starting with cell 1 and proceeding to cell 16. The computed outputs won't be available on the external I/Os until the start of the next evaluation cycle, but importantly the changed output of a cell can affect a later numbered cell in the same cycle. For example, if cell 2 uses the output of cell 1, there will be no apparent delay, but if cell 1 uses the output of cell 2 then it won't be updated until the next evaluation cycle.

The evaluation cycle is triggered by a clock signal. By default, the evaluation clock comes from an 4kHz source on the PLC. It is also possible for the clock to be provided by another Tiger card for synchronization purposes, e.g. for diSPIM the micro-mirror card is the source of the signals being output by the PLC (e.g. camera triggers) so the micro-mirror is the master and its 4kHz clock is used to trigger PLC evaluations. Finally, the user can provide a clock on the first BNC input. Note that there is an delay of one evaluation cycle because the outputs are updated at the start of the evaluation cycle; this is an intentional design choice so that the output timing is deterministic. Thus, an input signal with arbitrary phase relative to the evaluation clock is output with a delay between 1 and 2 times the period of the evaluation cycle clock.

Programming

The configuration of each cell can be changed by sending serial commands to the card. This can be done by either setting the entire card configuration to one of several presets or by sequentially setting the configuration of each cell. The configuration of the entire card is saved by executing a save settings operation, and need not be reprogrammed again until a different logic function is needed. The use of each physical connector is also user-programmed; each connector can be used as an digital input or mapped to reflect the output of any desired logic cell. Besides the 8 faceplate connectors, there are also 8 backplane connectors for inter-card communication in the controller (e.g. used in the diSPIM). [5)]

Programming Logic Cells

The logic cells start at index 1, to move the pointer position to cell 1 you would send the command `M E=1`.

To program the logic cells:

1. Move the axis position to the cell, e.g. (`M E=10`) specifies that the programming commands `CCA Y Z F` and `CCB` will be applied to the cell at position 10. The current pointer position can be queried using `W E`.
2. Execute `[Addr#]CCA Y=<code>` to select the type of cell. See first table below for cell types and codes. Setting the cell type (even to the exact same value) clears the previously-selected cell configuration, input selections, and state.
3. Execute `[Addr#]CCA Z=<code>` to set the cell configuration if applicable (see table). This is a 16-bit code that defines how the cell behaves, and its interpretation varies by cell type. Setting the cell configuration clears the state (only applies to one-shots and delay cells).
4. Execute `[Addr#]CCB X=<input #1> Y=<input #2> Z=<input #3> F=<input #4>` to set cell inputs, where the input address is from the second table. Inputs default to 0, or logic low. Note that the logical inversion of any address is easily obtained by simply adding 64 to the address. To obtain a rising or falling edge signal, add 128 or 192 to the address respectively. When an address is assigned to an edge-sensitive input (underlined in `Table: Cell Types` below), the firmware automatically converts the address to the range 128-255 if it is specified by the user in the range 0-127 (when such an input address is queried the value between 128 and 255 is returned). It is possible to combine the edge signals using combinatorial logic combine signals to feed to edge-sensitive inputs; e.g. to clock a flip flop when the output of cell 1 changes (either rising or falling edge) define a 2-input OR with input addresses 129 and 193 (193=1+128+64) and use the OR gate's output to clock the flip flop. [6)]

The remainder of this section discusses the available cell types.

Cell codes for use with "CCA" and "CCB" commands

Trigger and clock inputs are edge-sensitive and are underlined; all other inputs are level-sensitive. If the user sets an edge-sensitive input to address 0-127 then it will automatically be set to the corresponding edge-sensitive address 128-255. Address 192 (or 64) has a rising edge every evaluation cycle.

Table: Cell Types

Cell Type	Cell Type `CCA Y`	Configuration `CCA Z`	Input 1 `CCB X`	Input 2 `CCB Y`	Input 3 `CCB Z`	Input 4 `CCB F`	Has State?	Firmware
constant	0	0 for 0, 1 for 1	-	-	-	-	-	all
D-flop	1	-	Din	clock	reset	preset	Yes	all
2-input LUT	2	LUT values	A	B	-	-	-	all

3-input LUT	3	LUT values	A	B	C	-	-	all
4-input LUT	4	LUT values	A	B	C	D	-	all
AND2	5	-	A	B	-	-	-	all
OR2	6	-	A	B	-	-	-	all
XOR2	7	-	A	B	-	-	-	all
one-shot (retriggerable)	8	duration (# clock pulses)	trigger	clock	reset	-	Yes	all
delay (retriggerable)	9	delay to fire (# clock pulses)	trigger	clock	reset	-	Yes	all
AND4	10	-	A	B	C	D	-	all
OR4	11	-	A	B	C	D	-	all
Synchronous D-flop	12	-	Din	clock	reset(S)	preset(S)	Yes	all
JK-flop	13	-	J	K	clock	-	Yes	all
one-shot (non-retriggerable)	14	duration (# clock pulses)	trigger	clock	reset	-	Yes	all
delay (non-retriggerable)	15	delay to fire (# clock pulses)	trigger	clock	reset	-	Yes	all
one-shot OR2 (non-retriggerable)	16	duration (# clock pulses)	trigger A	clock	reset	trigger B	Yes	3.50+
delay OR2 (non-retriggerable)	17	delay to fire (# clock pulses)	trigger A	clock	reset	trigger B	Yes	3.50+
Async/sync D-flop	18	-	Din	clock	reset	reset(S)	Yes	3.51+
counter AND2	19	count (read-only)	A	clock	reset	B	Yes	not yet
counter OR2	20	count (read-only)	A	clock	reset	B	Yes	not yet
counter timer	21	count (read-only)	start	clock	reset	stop	Yes	not yet
counter timer (non-retriggerable)	22	count (read-only)	start	clock	reset	stop	Yes	not yet

Address codes (used with ''CCB'' command as well as setting source address for physical I/O)

Address	Meaning	Type
0	always 0/low	constant
1-16	logic cell 1-16	logic cell
17-32	reserved for future use	(expanded cell array)
33-40	front panel connectors 1-8	physical I/O
41-48	Tiger backplane TTL 0-7	physical I/O
49-63	reserved for future use	(expanded I/O)
64-127	0-63 with inversion	
128-191	rising edge on 0-63	
192-255	falling edge on 0-63	

Logic Gates

The predefined cell types include some common boolean logic gates for convenience, even though lookup tables could also be used. Two-input AND, OR, and XOR gates are provided, along with four-input AND and OR gates. A 3-input AND can be created using a 4-input cell and setting the unused input to address 64 (logic 1), and similarly a 3-input OR by setting the unused inputs to 0.

Lookup Tables (LUT)

Lookup tables (LUTs) are used to implement arbitrary boolean logic. In the case of a 4-input LUT, the eponymous lookup table code comprises 16 bits corresponding to the digital outputs of the 16 possible combinations of the 4 digital inputs. It is programed as the `CCA Z` code which ranges between 0 and 65535, where the LSB (bit 0) corresponds to all inputs 0 (logic low) and the MSB (bit 15) corresponds to all inputs 1 (logic high). Bit 1 of the lookup table code is the output when Input #1 is high and the others low, bit 2 of the code is the output when Input #2 is high and the others low, bit 3 is the output when Inputs #1 and #2 are high and the others are low, etc. The 16-bit lookup table code can be derived as follows:

input #1 (X)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
input #2 (Y)	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
input #3 (Z)	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
input #4 (F)	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
output	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
weight	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	64	32	16	8	4	2	1

First determine the desired output (0 or 1) in the penultimate row of the table. This is the binary form of the configuration code. To determine the decimal form used by the `CCA Z` command to program the cell, multiply each the output by the binary weight below and sum the values. For example, with a 4-input lookup table the operation ((X AND Y) OR (X AND Y AND Z AND F)) is represented by the CCA Z code of `1000100010001001` in boolean, or 34953 in decimal, and would be programmed by sending the command `CCA Z=34953`. The operation (Z OR F) is represented by the LUT `1111111111110000` in binary, or 65280 in decimal.

If only 2 or 3 inputs are required it is preferred to use the 2-input or 3-input LUT versions for slightly faster execution. For a 3-input LUT the input #4 (F) doesn't matter, and for a 2-input LUT inputs #3 and #4 (Z and F) don't matter, meaning the maximum `CCA Z` code would be 255 and 15 respectively.

Flip Flops

The D flip-flop is a simple latch or 1-bit memory. If its reset input is high then the output is unconditionally set low. If preset is high and reset is low, then the output is high. If both reset and preset are low (the normal state), then the output is the input at the time the last rising edge was seen on the clock input. The output is updated the same evaluation cycle as the clock, i.e. the D-to-Q delay is 0. The reset and preset only require an evaluation cycle to “take,” not a clock pulse; i.e. they act like asynchronous reset/presets.

The synchronous D-flop behaves just like the normal D-flop except the reset and preset inputs are only sampled on the rising clock edge, acting as synchronous reset/preset.

The async/sync D-flop behaves like a hybrid of the normal and synchronous D-flop: it has an asynchronous reset (input #3, behaves like input #3 of the normal D-flop) and a synchronous reset (input #4, behaves like input #3 of the synchronous D-flop). The output will be reset if input #3 is high regardless of the clock, and it will be reset if input #4 is high and the clock has a rising edge.

The low-level JK-flop is helpful to build some logic structures. It has two inputs, J and K along with a clock input. When the clock input has a rising edge, the output is updated according to its prior state and the J and K input values. The combination J=1 and K=0 sets the output to 1, the combination J=0 and K=1 sets the output to 0, the combination J=K=1 will toggle the output value, and J=K=0 will hold the current state.

One-shot

The output of a one-shot (also known as a monostable multivibrator) goes high for a specified duration upon receiving a rising edge on the trigger input. The output stays high until it has detected the user-specified number of rising edges on the clock input in later evaluation cycles (i.e. the clock input on the evaluation cycle when the trigger has a rising edge is ignored), and then its output goes low again. The number of clock cycles, or duration, is specified as the configuration for this cell using `CCA Z` (must be between 0 and 65535). Note that the clock signal is an input; if you wish it to be the same as the clock for the PLC evaluation cycle then set input #2 to address 192 (setting to 64 has the same result). If the duration is set to 0 then the output will never go high. If the duration is set to 1, then the one-shot output goes high on the trigger pulse and low again at the first rising edge on the clock input. The reset input resets the count and forces the output to be 0; it overrides the other inputs.

There are two varieties of one-shots. The standard one is retriggerable, meaning a trigger received while the output is high will cause the duration counter to be reset and the pulse length to be extended. In other words, the output will go low the specified number of clock cycles after the last-received trigger. For the non-retriggerable one-shot, the trigger input is ignored while the output is high, so the duration will always be the same unless it is reset.

There is a non-retriggerable “OR2” one-shot with a second trigger input. The two edge-sensitive input triggers are logically combined with an OR2 operation and the result is used as the trigger signal. This allows for a more cell-efficient implementation of schemes where two different signals may trigger the one-shot.

Delay

The delay cell is somewhat like the one-shot, except for its output goes high for a single clock cycle (not evaluation cycle [7]) some user-specified time after the initial trigger. The number of clock cycles, or duration, before the output goes high is set by the cell's configuration (`CCA Z`, must be between 0 and 65535) . There is a special case if the duration is set to 0: the output will go high immediately and low again on the next rising edge of the clock signal. If the duration is set to 1 then the output will go high on the next rising clock edge and low again on the second rising edge. With duration of 2 then the output goes high on the second rising edge and low again on the third edge, and so forth. If a delay and one-shot cell are both connected to the same trigger and clock signals and have the same configuration (duration) setting then the delay cell's output goes high the same evaluation cycle as the one-shot's output goes low again. Like the one-shot, the clock input is ignored on the evaluation cycle when the trigger's rising edge is received. Also like the one-shot, the reset input resets the count and forces the output to be 0; it overrides the other inputs.

A one-shot with delay can be created by combining a one-shot cell and delay cell, with the delay cell's output connected to the trigger signal of a one-shot cell, both running from the same clock.

The standard delay cell is retriggerable, meaning that the duration counter is reset by receipt of another trigger so the output pulse will come the specified amount of time after the last-received trigger. A non-retriggerable delay cell is also available, which ignores trigger pulses between the initial trigger and the output pulse.

There is a non-retriggerable OR2 delay cell with a second trigger input. The two edge-sensitive input triggers are logically combined with an OR2 operation and the result is used as the trigger signal. This allows for a more cell-efficient implementation of schemes where two different signals may trigger the delay.

Counter

Counter cells are useful for interrogating digital signals instead of generating them. The resulting count can be read out over serial using the CCA F command described here. There are several different variants of counter cells.

The internal count of the counter cell can have value from 0 to 65535. If the reset input is high then the internal count is set to 0, overriding any other inputs. Otherwise, on every rising edge of the clock input the internal count may be incremented depending on the values of the cell's two other inputs. Different counter variants have different conditions for incrementing the internal count. The output state of a counter cell is high whenever it is active.

For the AND2 counter, the cell is active and its output state is high whenever both input A and input B (inputs #1 and #4 respectively) have a logic high value. For each evaluation cycle where the counter cell is active, the internal count is incremented if there is a rising edge on the clock input. Note that if the count is not incremented if already at 65535.

For the OR2 counter, the cell is active and its output state is high whenever either input A or input B (inputs #1 and #4 respectively) have a logic high value. For each evaluation cycle where the counter cell is active, the internal count is incremented if there is a rising edge on the clock input. Note that if the count is not incremented if already at 65535.

For the timer counter, a rising edge on the start input activates cell and sets its output high. A rising edge on the stop input deactivates the counter and sets its output low. For each evaluation cycle where the counter is active (this includes the evaluation cycle when started but not the evaluation cycle when stopped), the internal count is incremented if there is a rising edge on the clock input. Note that if the count is not incremented if already at 65535.

The non-retriggerable timer counter is identical to the standard timer counter except that the start input only activates the cell if the internal count is equal to 0 (i.e. if the counter cell has been reset since the last time it was started).

To interrogate a single signal's high time, use the AND2 counter with one input connected to logic high (address 64) or the OR2 counter with one input connected to logic low (address 0).

To interrogate the high time of an individual address (e.g. a pulse duration), connect the rising and falling edges of that address to the start and stop inputs of a timer counter. Use a non-retriggerable timer counter to only interrogate the very first time that the signal goes high.

To interrogate the delay between two rising edges, connect the rising edges to the start and stop inputs of a (non-retriggerable) timer counter cell. Note that you must know which rising edge occurs first.

Configuring Physical I/Os

Physical I/Os include both the front panel BNCs and the TG-1000 backplane lines. Each has two associated registers containing the I/O type and the source address (from which address the output will come).

I/O type: input, open-drain output, or push-pull output

There are three types of I/Os: input, open-drain output, or push-pull output. By default the TG-1000 backplane I/Os are inputs and the front panel BNCs are push-pull outputs.[8)]

When an I/O is used as an output, it can either use the open-drain + pull-up resistor configuration or else a push-pull configuration.

For Rev A boards, the push-pull output logic levels are fixed at approximately 0.1V and 3.2V (the micro-controller supply voltage) and the source impedance is relatively low (typical value of 150ohm) but not low enough to drive an accessory with 50ohm input impedance. For Rev B boards, push-pull outputs go through a separate buffer which source at least 48mA, sufficient to drive 50 ohm TTL levels. Also on Rev B boards, the logic high voltage for push-pull is selectable between 5V (default) or 3.3V using a jumper on the PCB and the the low logic level is less than 0.1V. [9)]

Open-drain output mode can be used to allow different output logic levels but requires the addition of an external pull-up resistor. Most users will not need to use this mode on the BNC connectors, though it is used to communicate on the TG-1000 backplane. The logic high value should be no more than 5.5V. The maximum current sink is 100mA, but it is recommended to be much less (e.g. use 5V with a 5kohm pull-up and the current will be 10mA).

Each I/O has a pull-up or pull-down resistor on the printed circuit board to prevent the line from floating. This resistor acts as the input impedance when the I/O is used as an input.[10)] TG-1000 backplane connections have 10kohm pull-up resistors to 3.3V and are usually used in open-drain mode to communicate with other Tiger cards. The front panel BNCs each have a 27kohm resistor to ground (for Rev A boards the value is 10kohm). Finally, there is a series resistor between the BNC connectors and internal nodes for protection; the value is 100ohm for Rev A boards and 43ohm for Rev B. This series resistor makes it so the output voltage is not strictly 0 when the BNCs are used in open-drain output mode. For Rev B boards, the series resistor makes the push-pull output impedance approximately 50ohm for impedance matching.

To specify a I/O type, first move the axis pointer to the physical I/O address and then execute a `[Addr#]CCA Y` command setting Y to the I/O type code. The code is 0 for input, 1 for open-drain output, and 2 for push-pull output [11)]. For example, to set the first front panel connector to be a push-pull output, do `M E=33` followed by `[Addr#]CCA Y=2`. This is a separate but distinct use of the CCA Y command described in Section Programming Logic Cells; if the current axis position corresponds to a logic cell address then `CCA Y` sets the I/O type for the physical I/O, but if the axis position is on a logic cell then the cell type is set.

I/O type	**`CCA Y` code**	**Comments**
input	0	default for backplane
open-drain output	1	
push-pull output	2	default for front panel

Physical I/O source address

The source address setting does not matter if the I/O type is an input. If it is an output, the source address specifies from what programmable logic cell or other I/O the output will be taken from.

It is possible to assign the physical I/O source address to be another physical I/O. If the source is an input, note that the data will be sampled when the evaluation cycle is triggered, and also remember the fixed delay of a single clock cycle due to the evaluation cycle structure described in Section “Overview”.

To specify a source address, first move the axis pointer to the physical I/O address and then execute a `[Addr#]CCA Z` command setting Z to the source address.

For example, to set the front panel #1 connector to be the output of cell 2, send `M E=33` followed by `[Addr#]CCA Z=2`. The logical NOT of the signal is easily obtained by simply adding 64 to the source address, e.g. to output the logical complement of cell 2's output value send `[Addr#]CCA Z=66` after moving the axis pointer to the correct position. This is a separate but distinct use of the `CCA Z` command described in Section Programming Logic Cells, if the current axis position corresponds to a logic cell address then `CCA Z` sets the source address for the physical I/O, but if the axis position is on a logic cell then the cell configuration is set.

Available Presets

Card presets are programmed using the `[Addr#]CCA X=<preset #>` command. Depending on the preset, some or all of the cells will be changed (types, configuration, and input sources) as well as some of the I/Os (I/O types and source addresses). The cells and outputs not used in the preset are not affected. Thus, applying multiple presets can have a “cumulative” effect, as long as they don't overwrite each other. After setting a preset, the cells and outputs can be further modified.

The list of presets available is shown in the table below. Additional presets can be easily added upon request. BNC outputs are push-pull unless otherwise specified.

Preset	Description	Cells	BNC Outputs	Firmware	Usage / Notes
0	All cells set to constant 0	1-16	-	3.00+	
1	Simulate original SPIM TTL card	-	BNC1-4	3.00+	
2	Set cell 1 to constant 0	1	-	3.00+	diSPIM: not acquiring
3	Set cell 1 to constant 1	1	-	3.00+	diSPIM: acquiring
4	16-bit counter	1-16	-	3.00+	Clock input set to address 192, cell 1 is LSB toggle flop
5	BNC5 selected, turned on by cell 10	-	BNC5-8	3.04+	diSPIM: laser #1 selected
6	BNC6 selected, turned on by cell 10	-	BNC5-8	3.04+	diSPIM: laser #2 selected
7	BNC7 selected, turned on by cell 10	-	BNC5-8	3.04+	diSPIM: laser #3 selected
8	BNC8 selected, turned on by cell 10	-	BNC5-8	3.04+	diSPIM: laser #4 selected
9	BNC5-8 all turned off	-	BNC5-8	3.04+	diSPIM: no laser selected

Preset	Description	Cells	BNC Outputs	Firmware	Usage / Notes
10	Set cell 8 to constant 0	8	-	3.04+	diSPIM: laser disabled
11	Set cell 8 to constant 1	8	-	3.04+	diSPIM: laser enabled
12	Cell 10 as (TTL1 AND cell 8)	10	-	3.08+	diSPIM: selected laser on
13	BNC4 source as (TTL3 AND (cell 10 OR cell 1)	12	BNC4	3.06+	diSPIM: gated side output
14	diSPIM TTL	1, 8, 10, 12	BNC1-8	3.04+[12]	Combination of presets 1, 9, 12, 13
15	modulo 4 counter in cells 3 and 4, cell 2 as clock	3, 4	-	3.06+	diSPIM: counter for 4 lasers
16	modulo 3 counter in cells 3 and 4, cell 2 as clock	3, 4, 5	-	3.06+	diSPIM: counter for 3 lasers; note also uses cell 5
17	Cell 2 is !TTL1	2	-	3.06+	diSPIM: increment count every slice
18	Cell 2 is !TTL3	2	-	3.06+	diSPIM: increment count every volume (side A first)
19	BNC1-8 source from cells 9-16	-	BNC1-8	3.06+	Useful for testing together w/ preset 4
20	BNC5-8 source from cells 13-16	-	BNC5-8	3.07+	diSPIM: laser source from cells 13-16
21	modulo 2 counter in cells 3 and 4, cell 2 as clock (cell 4 always low)	3, 4	-	3.07+	diSPIM: counter for 2 lasers
22	no counter in cells 3 and 4 (both always low)	3, 4	-	3.07+	diSPIM: only 1 channel
23	BNC1-8 source from TTL0-7	-	BNC1-8	3.09+	
24	BNC3 source from cell 1	-	BNC3	3.09+	diSPIM: acquisition indicator
25	BNC3 source from cell 8	-	BNC3	3.09+	diSPIM: shutter indicator
26	Cell 2 is TTL3	2	-	3.10+	diSPIM: increment count every volume (side B first)
27	BNC3 source from cell 10	-	BNC3	3.17+	diSPIM: laser on indicator
28	BNC6 and BNC7 selected, turned on by cell 10	-	BNC5-8	3.19+	diSPIM: simultaneous 2-color
29	BNC5-7 selected, turned on by cell 10	-	BNC5-8	3.19+	diSPIM: simultaneous 3-color
30	BNC5-8 selected, turned on by cell 10	-	BNC5-8	3.19+	diSPIM: simultaneous 4-color
31	BNC5 and BNC7 selected on !TTL3, BNC6 and BNC8 on TTL3	6, 7	BNC5-8	3.19+	diSPIM: different 2 colors on side A and B (BNC5 and BNC7 for side A)
32	BNC1 and BNC2 set as cameras A and B		BNC1-2	3.19+	subset of preset 1
33	BNC1 and BNC2 set as OR of internal camera A and B signal	9	BNC1-2	3.19+	diSPIM: for reflective SPIM imaging
34	cell 11 as PLogic trigger divided by 2	11	-	3.23+	get output clock (divided by 2 from input clock)
35	BNC3 source from cell 11	-	BNC3	3.23+	output clock on BNC3
36	cell 10 set to reflect cell 8	10	-	3.27+	use shutter logic without micro-mirror TTL1 signal

Preset	Description	Cells	BNC Outputs	Firmware	Usage / Notes
37	BNC1 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #1 selected
38	BNC2 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #2 selected
39	BNC3 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #3 selected
40	BNC4 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #4 selected
41	BNC5 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #5 selected
42	BNC6 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #6 selected
43	BNC7 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: laser #7 selected
44	BNC2 and BNC4 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #2 and #4 selected
45	BNC3 and BNC5 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #3 and #5 selected
46	BNC4 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #4 and #6 selected
47	BNC5 and BNC7 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #4 and #6 selected
48	BNC1 and BNC3 and BNC5 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #1 and #3 and #5 selected
49	BNC2 and BNC4 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.29+	7 channel laser: lasers #2 and #4 and #6 selected
50	BNC1-7 all turned off	-	BNC1-7	3.29+	7 channel laser: all lasers off
51	BNC1-8 sourced from cells 17-24 (requires 24-cell build)	-	BNC1-8	3.29+	7 channel laser: no laser selected
52	BNC3 source from backplane TTL5 (stage sync trigger)	-	BNC3	3.31+	stage sync access
53	BNC1 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.35+	7 channel laser: lasers #1 and #6 selected
54	BNC1 and BNC4 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.35+	7 channel laser: lasers #1 and #4 and #6 selected
55	BNC1 and BNC4 selected, turned on by cell 10	-	BNC1-7	3.37+	7 channel laser: lasers #1 and #4 selected
56	BNC2 and BNC5 selected, turned on by cell 10	-	BNC1-7	3.37+	7 channel laser: lasers #2 and #5 selected
57	BNC3 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.37+	7 channel laser: lasers #3 and #6 selected
58	BNC1 and BNC5 selected, turned on by cell 10	-	BNC1-7	3.37+	7 channel laser: lasers #1 and #5 selected
59	BNC2 and BNC6 selected, turned on by cell 10	-	BNC1-7	3.37+	7 channel laser: lasers #2 and #6 selected
60	modulo 3 counter in cells 3 and 4, cell 2 as clock	3, 4	-	3.51+	diSPIM: counter for 3 lasers (replacement for preset 16)

Accessing outputs over serial

Most often the binary values of the programmable cells are used as digital signals on the physical connectors. However, the output of the programmable cells can also be read using a serial command. This may be useful, for example, if a counter has been created and high-level software needs to know the counter value. Likewise, the value of the physical I/Os can also be read. Users of these commands should understand that the values may be changing rapidly, and the commands only return a single-time snapshot of the state.

Use the `[Addr#]RDADC Z?` command to access the 16-bit unsigned integer corresponding to the output values of the 16 logic cells (addresses 1-16). The output of cell 1 is the LSB and the output of cell 16 is the MSB. As of v3.27 you can use the `[Addr#]RDADC F?` command to access the output values of the upper 16 logic cells (addresses 17-32, which cells are not present in the default builds).

`[Addr#]RDADC X?` returns the value of the front panel I/Os, with LSB corresponding to BNC #1. Floating inputs read as a logic low because board has a pull-down resistor to ground.

`[Addr#]RDADC Y?` returns the value of the Tiger backplane I/Os, with LSB corresponding to TTL0. Floating inputs read as logic high because the board has a pull-up resistor to 3.3V.

Accessing cell state over serial

Flip-flops, one-shots, delay cells, and counters all have state, which refers to the D-flop output (high or low), the current clock counter value in one-shot and delay elements (counter decreases with each clock), and the count in counter cells. Generally the logic cell state is not manipulated directly but there is a mechanism to do so if required. Note that cell state is not the same as the binary output value which can be accessed using `[Addr#]RDADC Z?`.

Use the `HOME <axis>` or `! <axis>` command to clear the state of all cells, e.g. `! E`.

The command `CCA F` can be used to read or write the state of the currently-selected cell according to the axis pointer. The state of flip-flops has value either 0 or 1. The state of one-shots and delay cells is the internal counter if active (the counter decreases until 0 with successive clocks) or 0 if the one-shot or delay isn't currently triggered. For example, if cell 11 contains a D-flop then its state (0 or 1) can be read by first executing `M E=11` followed by `[Addr#]CCA F?`. Likewise, the D-flop state (and hence output value) could be set high by executing `M E=11` followed by `[Addr#]CCA F=1`.

Changing trigger source

By default the evaluation cycle happens on an internal 4kHz clock. However, the evaluation cycle can be triggered by hardware interrupt instead. For diSPIM, the micro-mirror card generates trigger signals for cameras and lasers which are controlled by PLC outputs. To maintain synchronization, the PLC trigger source is set to be the 4kHz clock of the micro-mirror card.

There is an option for the user to provide a clock for the evaluation cycle on the first BNC front panel connector. When changing to this trigger source, BNC input 1 is set to be an input automatically and cannot be changed to an output until the trigger source has been changed.

To change trigger mode, use the PM command, which is axis-specific, to change the trigger source as desired. For example, PM E=1 would be used for diSPIM with the micro-mirror card master clock is on backplane C7. The table below contains a list of trigger source codes.

Code	Trigger source	Notes
0	internal 4kHz clock	default
1	Backplane C7	for diSPIM, from micro-mirror card (P3.0)
2	Backplane C13 (TTL5)	
3	Backplane C14 (TTL7)	
4	BNC input 1	

Save card settings

Like other cards, the save settings command (`[Addr#]SS Z`) can be sent to the programmable logic card to save all settings in nonvolatile memory where it will be restored when the controller is powered on the next time. Cell states (e.g. flop values) are not remembered, but the following settings are remembered using SS Z:

- logic cell types, configuration, and input assignments
- I/O types and source addresses
- card trigger source

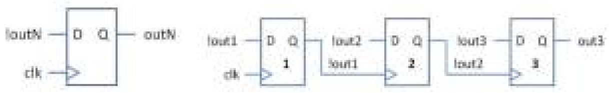
Example Logic

Toggle

A toggle output changes output state once per clock input, making it a divide-by-2. There are multiple ways of implementing a toggle using the available logic structures. One is to use a D-flop with input connected to the inverted version of its own output. Another is to use a 2-input XOR with one input connected to logic high (address 64) and the other connected to its own output. Finally, you can use a 2-input OR with one input connected to logic low (address 0) and the other connected to the inverted version of its own output.

Simple counter

A simple N-bit counter consists of N cascaded toggle flops. Each toggle flop is simple in structure; a counter is formed by cascading toggle flops, with the output of each cell acting as the clock for the following cell. The figure below shows this for a 3-bit up counter. This is knows as a “ripple counter” because the lower order bit has to toggle before the next bit can flip. In digital logic there is a transient time where the observed count will be inaccurate while the outputs are changing. However, in the PLC, as long as each higher-order toggle flop is in a higher-number cell then the counter will act as a synchronous counter because of the sequential evaluation of cells' logic value without any output until the next evaluation cycle. To make a down counter, invert the polarity of the clock inputs.



Clock

To create a free-running clock with user-selectable period and duty cycle you can combine two non-retriggerable one-shot cells. The first cell's trigger is connected to logic high, but because it is non-retriggerable it will simply trigger itself whenever it has, thus outputting a pulse once per delay period. That pulse triggers a second one-shot which generates the clock output whose duration sets the time the output signal is high (i.e. the duty cycle is the ratio of the one-shot duration with the delay period). From the 4kHz internal clock it is possible to generate up to 2kHz clock by this means.

Script: clock signal on BNC3 output

clock.bsh

```
// generate a clock signal based on the internal 4kHz clock, e.g.
for triggering a camera
// currently hard-codes using cells 1 and 2 and BNC output #3

// variables to be edited by user
double clockFrequencyHz = 100;  // clock frequency in Hertz
double clockDutyCycle = 0.5;     // should be between 0 and 1
exclusive

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
String valOneShotNRT = "14 - one shot (NRT)";
int addrInvert= 64;
int addrEdge = 128;
double ticsPerSecond = 4000.0;
int addrOutputBNC3 = 35;
int addrDelayNRT = 1;
int addrOneShot = 2;

// figure out the cycle period and high period in terms of PLC
"tics" (4kHz)
int clockPeriodTics = (int) (ticsPerSecond/clockFrequencyHz +
0.5);
print("have " + (clockPeriodTics-1) + " (+1) tics in one-shot for
actual frequency of " + ticsPerSecond/(clockPeriodTics));
int clockHighTics = (int) (clockPeriodTics * clockDutyCycle);

// turn off updates to speed communication
```

```
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// do programming of one-shot and delay cells
mmc.setProperty(plcName, propPosition, addrDelayNRT);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, (clockPeriodTics-1));
mmc.setProperty(plcName, propCellInput1, addrInvert + addrEdge);
// trigger cell as fast as we can, but NRT prevents re-triggering
when active
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic

mmc.setProperty(plcName, propPosition, addrOneShot);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, clockHighTics);
mmc.setProperty(plcName, propCellInput1, addrDelayNRT);  //
trigger cell as fast as we can, but NRT prevents re-triggering
when active
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic

// connect to BNC output
mmc.setProperty(plcName, propPosition, addrOutputBNC3);
mmc.setProperty(plcName, propCellConfig, addrOneShot);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

Fixed number of pulses from trigger

A single trigger signal can be used to generate a train of pulses with fixed period. This might be useful to take the stage synchronization pulse and generate a series of camera triggers.

Script: fixed number of pulses after initial trigger

npulses.bsh

```
// Generate a user-specified number of pulses with user-specified
period and duration
// upon receipt of a single rising edge trigger.  The trigger here
is the stage sync signal.

// variables to be edited by user
int numPulses = 25;           // number of pulses to send
double pulsePeriodMs = 10.0; // period of pulses in milliseconds
double pulseHighMs = 1.0;    // time in milliseconds that the
```

```
pulse is high for, should be less than pulse period
int addrOutputBNC1 = 33;     // address 33 is BNC#1 on PLC front
panel
int addrStageSync = 46;      // address 46 is TTL5 on Tiger
backplane = stage sync signal

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
String valDFlop = "1 - D flop";
String valOneShotNRT = "14 - one shot (NRT)";
int addrZero = 0;
int addrInvert = 64;
int addrEdge = 128;
double ticsPerMs = 4.0;
int addrActive = 1;
int addrPeriod = 2;
int addrPulse = 3;
int addrCounter = 4;

// figure out the cycle period and high period in terms of PLC
"tics" (4kHz)
int pulsePeriodTics = (int) (ticsPerMs * pulsePeriodMs + 0.5);
int pulseHighTics = (int) (ticsPerMs * pulseHighMs + 0.5);

// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// force BNC output to be low so no transients seen
mmc.setProperty(plcName, propPosition, addrOutputBNC1);
mmc.setProperty(plcName, propCellConfig, addrZero);

// D-flop registers when master trigger signal (from stage)
arrives
// we reset this once we have output the desired number of pulses
mmc.setProperty(plcName, propPosition, addrActive);
mmc.setProperty(plcName, propCellType, valDFlop);
mmc.setProperty(plcName, propCellInput1, addrInvert);  // connect
D input to logic 1
mmc.setProperty(plcName, propCellInput2, addrEdge +
addrStageSync);  // trigger/clock with stage
```

```
mmc.setProperty(plcName, propCellInput3, addrEdge + addrInvert +
addrCounter);  // reset once counter is done

// one-shot NRT to keep period, clocked every tic and immediately
set
//    (but initially kept reset until D-flop captures pulse)
mmc.setProperty(plcName, propPosition, addrPeriod);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, pulsePeriodTics - 1);
mmc.setProperty(plcName, propCellInput1, addrInvert + addrEdge);
// trigger every time
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock at 4kHz
mmc.setProperty(plcName, propCellInput3, addrInvert + addrActive);
// keep reset unless we got triggered

// output pulse of the desired duration
mmc.setProperty(plcName, propPosition, addrPulse);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, pulseHighTics);
mmc.setProperty(plcName, propCellInput1, addrEdge + addrPeriod);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock at 4kHz

// one-shot to track how many pulses we have remaining
mmc.setProperty(plcName, propPosition, addrCounter);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, numPulses-1);  // I don't
fully understand why the -1 is needed ;-)
mmc.setProperty(plcName, propCellInput1, addrStageSync);
mmc.setProperty(plcName, propCellInput2, addrEdge + addrPulse);

// connect to BNC output
mmc.setProperty(plcName, propPosition, addrOutputBNC1);
mmc.setProperty(plcName, propCellConfig, addrPulse);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

Pass through pulse N*M times

A customer had an external pulse that they wanted to pass through a set number of times and then disable the pass-through. This script implements a large counter by stringing together two one-shot cells acting as counters – thus allowing the total number of pulses to be passed through to be a product of two numbers that each can be as large as 65535. There is also a one-shot to latch the first time the second one-shot rolls over and an additional flop to delay that so that the final pulse makes it through the final AND gate. This script initializes everything when it runs, or alternatively set the cell 1 to a logic high and then logic low again.

Script: Pass through pulse N*M times

passthrough.bsh

```
// Pass through pulses until the amount seen exceeds a user-set
number.
// Number of pulses to pass through is specified by the product of
two values each of which can be 65k (so total can be ~2^32)
// Initialize via serial command (including running this script,
or setting cell address 1 from 0 to 64 (low to high))

// variables to be edited by user
int numPulsesInner = 3;     // number of pulses to send, up to
2^16-1
int numPulsesOuter = 7;     // number of pulses to send, up to
2^16-1
int addrPulseIn = 33;        // address 33 is BNC#1 on PLC front
panel
int addrPulseOut = 40;       // address 40 is BNC#8 on PLC front
panel

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
String valConstant = "0 - constant";
String valDFlop = "1 - D flop";
String valAND2 = "5 - 2-input AND";
String valOneShotNRT = "14 - one shot (NRT)";
String valIOInput = "0 - input";
String valIOPushPull = "2 - output (push-pull)";
int addrZero = 0;
int addrHigh = 64;
int addrInvert = 64;
int addrEdge = 128;
// evaluation order of these is important
int addrInitialize = 1;
int addrInnerCount = 2;
int addrOuterCount = 3;
int addrLatchFlop = 4;
int addrDelayFlop = 5;
int addrOutputAnd = 6;


// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// first one-shot to track how many pulses we have remaining in
this round
// clock is from external pulse but trigger is via serial on cell
1
mmc.setProperty(plcName, propPosition, addrInnerCount);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, numPulsesInner - 1);
mmc.setProperty(plcName, propCellInput1, addrEdge + addrPulseIn);
mmc.setProperty(plcName, propCellInput2, addrEdge + addrPulseIn);
mmc.setProperty(plcName, propCellInput3, addrInitialize);

// second one-shot to track how many rounds we have remaining
// clock is from first one-shot but trigger is via serial on cell
1
mmc.setProperty(plcName, propPosition, addrOuterCount);
mmc.setProperty(plcName, propCellType, valOneShotNRT);
mmc.setProperty(plcName, propCellConfig, numPulsesOuter - 1);
mmc.setProperty(plcName, propCellInput1, addrEdge + addrInvert +
addrInnerCount);
mmc.setProperty(plcName, propCellInput2, addrEdge + addrInvert +
addrInnerCount);
mmc.setProperty(plcName, propCellInput3, addrInitialize);

// D-flop registers when final count has been reached
mmc.setProperty(plcName, propPosition, addrLatchFlop);
mmc.setProperty(plcName, propCellType, valDFlop);
mmc.setProperty(plcName, propCellInput1, addrHigh);  // connect D
input to logic 1
mmc.setProperty(plcName, propCellInput2, addrEdge + addrInvert +
addrOuterCount);
mmc.setProperty(plcName, propCellInput3, addrInitialize);  //
reset

// D-flop delay to wait for the final pulse to go through
mmc.setProperty(plcName, propPosition, addrDelayFlop);
mmc.setProperty(plcName, propCellType, valDFlop);
mmc.setProperty(plcName, propCellInput1, addrLatchFlop);  //
connect D input to latch flop
mmc.setProperty(plcName, propCellInput2, addrEdge + addrInvert +
addrPulseIn);    // clock on falling edge of pulse signal
mmc.setProperty(plcName, propCellInput3, addrInitialize);  //
reset

// AND gate to pass through pulse to output if the count hasn't
```

```
been exceeded
mmc.setProperty(plcName, propPosition, addrOutputAnd);
mmc.setProperty(plcName, propCellType, valAND2);
mmc.setProperty(plcName, propCellInput1, addrPulseIn);
mmc.setProperty(plcName, propCellInput2, addrDelayFlop +
addrInvert);

// connect output pulse to BNC output (and ensure it is an output)
mmc.setProperty(plcName, propPosition, addrPulseOut);
mmc.setProperty(plcName, propCellType, valIOPushPull);
mmc.setProperty(plcName, propCellConfig, addrOutputAnd);

// ensure that the BNC input is set as such
mmc.setProperty(plcName, propPosition, addrPulseIn);
mmc.setProperty(plcName, propCellType, valIOInput);

// initialize the one-shot counters by doing the initial trigger
mmc.setProperty(plcName, propPosition, addrInitialize);
mmc.setProperty(plcName, propCellType, valConstant);
mmc.setProperty(plcName, propCellConfig, addrZero);
mmc.setProperty(plcName, propCellConfig, addrHigh);
mmc.setProperty(plcName, propCellConfig, addrZero);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

Output the stage sync signal

For stage scanning there is an internal signal generated when the stage is scanning, and commonly the rising edge can be used to trigger other logic (e.g. camera that is free-running during the stage scan). This script configures the internal signal to be immediately output on BNC #3 of the PLC. Note that this stage sync signal requires a motorized 2-axis card with an appropriate jumper in place and also firmware with the `SCAN_MODULE` firmware (some documentation at SCAN MODULE but needs to be augmented). [13)]

Script: Stage sync signal on BNC

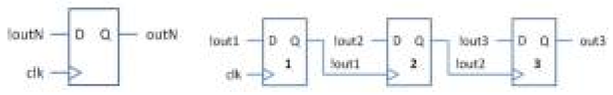
stagesync.bsh

```
// output the internal stage sync signal on BNC output #3
// stage sync signal goes high when the stage passes the start
position and falls when it passes the end position
// (set up start and stop positions using SCANR command X and Y
parameters, then use SCAN command to initiate)

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellConfig = "EditCellConfig";
int addrOutputBNC3 = 35;
int addrStageSync = 46;  // TTL5 on Tiger backplane = stage sync 
signal

// connect to BNC output
mmc.setProperty(plcName, propPosition, addrOutputBNC3);
mmc.setProperty(plcName, propCellConfig, addrStageSync);
```

diSPIM 2 laser control

Say we want to use two lasers with diSPIM where the laser color is changed on alternate volumes. The micro-mirror drive card of the Tiger outputs are a single laser trigger and a TTL side indicator. For two lasers where the laser is toggled ever volume, we to route the laser trigger to one laser for the first volume and the second laser on the next volume. This is accomplished by using a toggle flop (1-bit counter) that is toggled after every volume. This requires 3 logic cells, arranged as shown here. The BNC connectors on the front of the PLC provide the required additional physical connectors.

If instead we want to toggle the laser every slice, the logic is almost identical except that the toggle counter is connected to the laser trigger. Some additional logic could be added to reset the counter between stacks in case an odd number of slices is used.



Micro-Manager ASIdiSPIM plugin

The Micro-Manager ASIdiSPIM plugin uses the PLC for outputting the camera and laser triggers. Here is the usage of the internal cells:

Cell	**Use**
1	acquisition flag
2	laser counter clock source
3	laser counter
4	laser counter
5	laser counter (needed for 3 channels only)
6	
7	
8	shutter flag
9	
10	laser on

Cell	Use
11	
12	side select switch
13	BNC5 laser
14	BNC6 laser
15	BNC7 laser
16	BNC8 laser

diSPIM pulse after acquisition

An end user wanted to trigger another piece of equipment a certain time after the conclusion of the diSPIM acquisition in Micro-Manager. We use the acquisition flag that the Micro-Manager plugin sets high before acquisition and low after it is completed (there may be a variable lag of 0.3-0.8 second from when the hardware is actually finished). We use that flag to clock a delay cell which in turn controls a one-shot to keep the trigger output high for a specified duration. This script saves the settings to the controller so it should only need to be run once unless the delay is changed, firmware is updated, or another script using cells 6 or 7 or output #3 is run.

Script: diSPIM pulse after acquisition

pulseafteracq.bsh

```
// variables to be edited by user
delayInSeconds = 2;     // can be no more than 16.3
durationInSeconds = 1;  // can be no more than 16.3

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propUpdates = "EditCellUpdateAutomatically";
String propSaveSettings = "SaveCardSettings";
String valNo = "No";
String valDelay = "15 - delay (NRT)";
String valOneShot = "14 - one shot (NRT)";
String valSaveSettings = "Z - save settings to card (partial)";
int addrAcqFlag = 1;
int addrDelayCell = 6;
int addrOneShotCell = 7;
int addrOutput3 = 35;
int addrInvert= 64;
int addrEdge = 128;
int ticsPerSecond = 4000;
```

```
// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// program cell 6 as delay with user-specified value and triggered
every clock
mmc.setProperty(plcName, propPosition, addrDelayCell);
mmc.setProperty(plcName, propCellType, valDelay);
mmc.setProperty(plcName, propCellConfig,
ticsPerSecond*delayInSeconds);
mmc.setProperty(plcName, propCellInput1, addrAcqFlag + addrInvert
+ addrEdge);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic
mmc.setProperty(plcName, propCellInput3, addrAcqFlag + addrEdge);
// reset when acquisition starts

// program cell 7 as a one-shot
mmc.setProperty(plcName, propPosition, addrOneShotCell);
mmc.setProperty(plcName, propCellType, valOneShot);
mmc.setProperty(plcName, propCellConfig,
ticsPerSecond*durationInSeconds);
mmc.setProperty(plcName, propCellInput1, addrDelayCell);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic
mmc.setProperty(plcName, propCellInput3, addrAcqFlag + addrEdge);
// reset when acquisition starts

// output #3 reflects cell 6
mmc.setProperty(plcName, propPosition, addrOutput3);
mmc.setProperty(plcName, propCellConfig, addrOneShotCell);

// save settings
mmc.setProperty(plcName, propSaveSettings, valSaveSettings);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

iSPIM pulse during acquisition

An end user wanted to trigger another piece of equipment in the middle of a single-view (iSPIM) acquisition in Micro-Manager. This trigger needed to occur a certain time after acquisition began, have a specified duration, and not occur again until the next acquisition. The strategy is to use the Camera A trigger to initiate a delay element which in turn triggers a one-shot. The initial logic element is a D-flop to generate a rising edge on its output one time only until it is reset by the acquisition flag. This script saves the settings to the controller so it should only need to be run once unless the delay is changed, firmware is updated, or another script using cells 6 or 7 or 9 or output #3 is run.

Script: iSPIM pulse during acquisition

ispim_pulseduringacq.bsh

```
// variables to be edited by user
double delayInSeconds = 3.0;     // can be no more than 16.3
double durationInSeconds = 1.0;  // can be no more than 16.3

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propUpdates = "EditCellUpdateAutomatically";
String propSaveSettings = "SaveCardSettings";
String valNo = "No";
String valDFlop = "1 - D flop";
String valDelay = "15 - delay (NRT)";
String valOneShot = "14 - one shot (NRT)";
String valSaveSettings = "Z - save settings to card (partial)";
int addrAcqFlag = 1;
int addrOnceCell = 6;
int addrDelayCell = 7;
int addrOneShotCell = 9;
int addrOutput3 = 35;
int addrCameraA = 41;
int addrInvert= 64;
int addrEdge = 128;
int ticsPerSecond = 4000;

// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// program cell 6 as a one-time-triggered flop that stays high
until reset when acquition starts
mmc.setProperty(plcName, propPosition, addrOnceCell);
mmc.setProperty(plcName, propCellType, valDFlop);
mmc.setProperty(plcName, propCellInput1, addrInvert);   // input
is logic 1
mmc.setProperty(plcName, propCellInput2, addrCameraA);  // clock
with CamA
mmc.setProperty(plcName, propCellInput3, addrAcqFlag + addrEdge);
// reset when acquisition starts
// program cell 7 as delay with user-specified value
mmc.setProperty(plcName, propPosition, addrDelayCell);
mmc.setProperty(plcName, propCellType, valDelay);
mmc.setProperty(plcName, propCellConfig,
(int)(ticsPerSecond*delayInSeconds));
mmc.setProperty(plcName, propCellInput1, addrOnceCell + addrEdge);
// trigrer when other flop goes high
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic
mmc.setProperty(plcName, propCellInput3, addrAcqFlag + addrEdge);
// reset when acquisition starts

// program cell 9 as a one-shot to stay high the user-specified
duration
mmc.setProperty(plcName, propPosition, addrOneShotCell);
mmc.setProperty(plcName, propCellType, valOneShot);
mmc.setProperty(plcName, propCellConfig,
(int)(ticsPerSecond*durationInSeconds));
mmc.setProperty(plcName, propCellInput1, addrDelayCell);
// trigger when delay is finished
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic
mmc.setProperty(plcName, propCellInput3, addrAcqFlag + addrEdge);
// reset when acquisition starts

// output #3 reflects cell 9
mmc.setProperty(plcName, propPosition, addrOutput3);
mmc.setProperty(plcName, propCellConfig, addrOneShotCell);

// save settings
mmc.setProperty(plcName, propSaveSettings, valSaveSettings);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

diSPIM multiple lasers

An end user wanted to illuminate one one side of the diSPIM with 2 out of the 4 laser lines (both lasers on simultaneously) and then use the other two lasers for illuminating on the second side. First we use two cells as 4-input AND gates with proper inputs to produce “laser on” signals at the appropriate times. Then we make the 4 laser output connectors reflect the binary values of those AND gates.

Here is the Beanshell script to do this using the Micro-Manager device adapter's properties. **Note that as of firmware v3.19 this script is not needed; simply use preset 31**.

Script: diSPIM multiple lasers

dispim_multilaser.bsh

```
// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellCellConfig";
String valAND4 = "10 - 4-input AND";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propCellInput4 = "EditCellInput4";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
int addrLaserOn = 42;
int addrLaserSide = 44;
int addrInvert = 64;
int addrSideAGate = 6;
int addrSideBGate = 7;
int addrShutterOpen = 8;
int addrOutput5 = 37;
int addrOutput6 = 38;
int addrOutput7 = 39;
int addrOutput8 = 40;

// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// do programming of 2 logic cells for two laser on/off signals
mmc.setProperty(plcName, propPosition, addrSideAGate);
mmc.setProperty(plcName, propCellType, valAND4);
mmc.setProperty(plcName, propCellInput1, addrLaserOn);
mmc.setProperty(plcName, propCellInput2, addrShutterOpen);
mmc.setProperty(plcName, propCellInput3, addrLaserSide +
addrInvert);
mmc.setProperty(plcName, propCellInput4, addrInvert);

mmc.setProperty(plcName, propPosition, addrSideBGate);
mmc.setProperty(plcName, propCellType, valAND4);
mmc.setProperty(plcName, propCellInput1, addrLaserOn);
mmc.setProperty(plcName, propCellInput2, addrShutterOpen);
mmc.setProperty(plcName, propCellInput3, addrLaserSide);
mmc.setProperty(plcName, propCellInput4, addrInvert);

// do programming of laser outputs
mmc.setProperty(plcName, propPosition, addrOutput5);
mmc.setProperty(plcName, propCellConfig, addrSideAGate);
mmc.setProperty(plcName, propPosition, addrOutput6);
mmc.setProperty(plcName, propCellConfig, addrSideBGate);
mmc.setProperty(plcName, propPosition, addrOutput7);
mmc.setProperty(plcName, propCellConfig, addrSideAGate);
mmc.setProperty(plcName, propPosition, addrOutput8);
mmc.setProperty(plcName, propCellConfig, addrSideBGate);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

SPIM reduce sheet width for portion of stack

We had a customer imaging zebrafish and wanting to avoid directing the laser into their eyes, so they wanted to narrow the width of the light sheet for a particular section of the z-stack. This was implemented on the PLC by selectively gating some laser-on pulses.

Here is a

PDF

with the basic concept. Below is the Beanshell script for use in Micro-Manager. It uses cells 6, 7, 9, and 11 additional to what are typically used by Micro-Manager plugin. The script makes the simplifying assumption that you aren't doing multiple timepoints, multiple positions, or multiple channels. If any of those are true then it will need to be tweaked to appropriately reset the delay and one-shot cells that figure out when the “special” time is.

Script: reduce sheet width for portion of stack

reducesheetwidth.bsh

```
// these 4 values are intended to be edited by the user
int slicesBeforeSpecial = 2;   // how many slices before starting
the shorter scan region
int slicesSpecial = 1;         // how many slices in the eye
region with the shorter scan
double msCutFromSlice = 0;  // how many ms to cut from the first
part of the scan
double msShortSlice = 3;        // how many ms to leave the laser
on for the shorter scan region

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String valLUT3 = "3 - 3-input LUT";
String valOneshotNRT = "14 - one shot (NRT)";
String valDelayNRT = "15 - delay (NRT)";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propCellInput4 = "EditCellInput4";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
int addrTTL1 = 42;
int addrAcq = 1;
int addrInvert = 64;
int addrEdge = 128;
int addrDelayGate = 6;
int addrGate = 7;
int addrDelayEyes = 9;
int addrEyes = 11;
int addrSlicesTrig = 16;
int addrLaser = 10;
int laserLookupTable = 216;

int ticsBeforeGate = Math.round(msCutFromSlice*4);
int ticsGate = Math.round(msShortSlice*4);

// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// do programming of 5 logic cells
mmc.setProperty(plcName, propPosition, addrDelayEyes);
mmc.setProperty(plcName, propCellType, valDelayNRT);
mmc.setProperty(plcName, propCellConfig, slicesBeforeSpecial);
mmc.setProperty(plcName, propCellInput1, addrAcq + addrEdge);
mmc.setProperty(plcName, propCellInput2, addrTTL1 + addrInvert +
addrEdge);
mmc.setProperty(plcName, propCellInput3, addrAcq + addrInvert);

mmc.setProperty(plcName, propPosition, addrEyes);
mmc.setProperty(plcName, propCellType, valOneshotNRT);
mmc.setProperty(plcName, propCellConfig, slicesSpecial);
mmc.setProperty(plcName, propCellInput1, addrDelayEyes);
mmc.setProperty(plcName, propCellInput2, addrTTL1 + addrInvert +
addrEdge);
mmc.setProperty(plcName, propCellInput3, addrAcq + addrInvert);

mmc.setProperty(plcName, propPosition, addrDelayGate);
mmc.setProperty(plcName, propCellType, valDelayNRT);
mmc.setProperty(plcName, propCellConfig, ticsBeforeGate);
mmc.setProperty(plcName, propCellInput1, addrTTL1);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
mmc.setProperty(plcName, propCellInput3, addrTTL1 + addrInvert +
addrEdge);
```

```
mmc.setProperty(plcName, propPosition, addrGate);
mmc.setProperty(plcName, propCellType, valOneshotNRT);
mmc.setProperty(plcName, propCellConfig, ticsGate);
mmc.setProperty(plcName, propCellInput1, addrDelayGate);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
mmc.setProperty(plcName, propCellInput3, addrTTL1 + addrInvert +
addrEdge);

mmc.setProperty(plcName, propPosition, addrLaser);
mmc.setProperty(plcName, propCellType, valLUT3);
mmc.setProperty(plcName, propCellConfig, laserLookupTable);
mmc.setProperty(plcName, propCellInput1, addrEyes);
mmc.setProperty(plcName, propCellInput2, addrGate);
mmc.setProperty(plcName, propCellInput3, addrTTL1);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

Constant-value Output

One customer wanted to set the value of output #3 to be continuously high for the experiment.

Script: output 3 always high

output3high.bsh

```
// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellConfig = "EditCellConfig";
String propSaveSettings = "SaveCardSettings";
String valSaveSettings = "Z - save settings to card (partial)";
int addrOutput3 = 35;
int addrHigh= 64;
int addrLow = 0;


// output #3 assigned to be always high
mmc.setProperty(plcName, propPosition, addrOutput3);
mmc.setProperty(plcName, propCellConfig, addrHigh);  // output to
be high
//mmc.setProperty(plcName, propCellConfig, addrLow);  // output to
be low
```

```
// save settings
mmc.setProperty(plcName, propSaveSettings, valSaveSettings);
```

Conditional Synchronized Output Pulse

One customer wanted to trigger a laser for a set time synchronized to the camera trigger signal, but only if a condition had been met (specifically a trigger on the controller backplane indicating the laser should be fired).

Script: output timed pulse synchronized to input

inoutsync.bsh

```
// output a timed pulse coincident with rising edge of input#1 if and only if input#2 has seen a rising edge since the last pulse
// use to synchronize the output with input#1
// currently set to output BNC#3 with input#1 of BNC#1 and input#2 of backplane TTL1
// example use case: fire laser (output) when the camera trigger is just starting (input#1) if we are ready for it (input#2)

// variables to be edited by user
outputDurationInMs = 25;  // can be no more than 16.3 seconds with normal 4kHz clock
int addrOutput = 35;  // 35 corresponds to BNC#3, change if want different output
int addrInput1 = 33;   // 33 corresponds to BNC#1, change if want different input
int addrInput2 = 42;   // 42 corresponds to TTL1 on backplane = laser shutter signal

// variables that should not need to be edited by user
// would like to define these variables as final, but this is
//  not amenable to script which can run multiple times
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propUpdates = "EditCellUpdateAutomatically";
String valNo = "No";
String valDFlop = "1 - D flop";
String valAND2 = "5 - 2-input AND";
String valOneShot = "14 - one shot (NRT)";
String valIOInput = "0 - input";
String valIOPushPull = "2 - output (push-pull)";
int ticsPerMs = 4;
int addrInvert= 64;
int addrEdge = 128;
int addrInput2Capture = 1;
int addrBothInputs = 2;
int addrOneShotCell = 3;


// figure out the high period in terms of PLC "tics" (4kHz)
int outputDurationTics = (int) (outputDurationInMs*ticsPerMs +
0.5);

// turn off updates to speed communication
String valUpdatesOriginal = mmc.getProperty(plcName, propUpdates);
mmc.setProperty(plcName, propUpdates, valNo);

// use D-flops to latch rising edges of two inputs
// (reset once output pulse starts)
mmc.setProperty(plcName, propPosition, addrInput2Capture);
mmc.setProperty(plcName, propCellType, valDFlop);
mmc.setProperty(plcName, propCellInput1, addrInvert);  // connect
D input to logic 1
mmc.setProperty(plcName, propCellInput2, addrEdge + addrInput2);
mmc.setProperty(plcName, propCellInput3, addrEdge +
addrOneShotCell);  // reset once output pulse starts

// use AND to combine edge of input#1 with latched signal from
input#2
mmc.setProperty(plcName, propPosition, addrBothInputs);
mmc.setProperty(plcName, propCellType, valAND2);
mmc.setProperty(plcName, propCellInput1, addrEdge + addrInput1);
mmc.setProperty(plcName, propCellInput2, addrInput2Capture);

// use non-retriggerable one-shot to output pulse
mmc.setProperty(plcName, propPosition, addrOneShotCell);
mmc.setProperty(plcName, propCellType, valOneShot);
mmc.setProperty(plcName, propCellConfig, outputDurationTics);
mmc.setProperty(plcName, propCellInput1, addrBothInputs);
mmc.setProperty(plcName, propCellInput2, addrInvert + addrEdge);
// clock on every tic

// make sure inputs are set as inputs
mmc.setProperty(plcName, propPosition, addrInput1);
mmc.setProperty(plcName, propCellType, valIOInput);
mmc.setProperty(plcName, propPosition, addrInput2);
mmc.setProperty(plcName, propCellType, valIOInput);

// connect one shot to output and set as output
mmc.setProperty(plcName, propPosition, addrOutput);
mmc.setProperty(plcName, propCellType, valIOPushPull);
mmc.setProperty(plcName, propCellConfig, addrOneShotCell);

// restore updates
mmc.setProperty(plcName, propUpdates, valUpdatesOriginal);
```

Blinking outputs for testing

To check functionality of the outputs and receipt of the input trigger signal it is convenient to set the 8 outputs corresponding to a digital counter which is advanced every evaluation cycle. Because the evaluation cycle is nominally 4 kHz visual inspection is easiest if only the highest bits of the counter are displayed. This is accomplished by sending the following two serial commands to re-configure the cells to a useful test mode:

`6 CCA X=4` (sets cells 1-16 as a digital counter advanced each evaluation cycle)

`6 CCA X=19` (sets BNC outputs 1-8 to correspond to cells 9-16)

Print internal configuration

The script below outputs the internal configuration of the PLC

Script: print internal configuration

printconfig.bsh

```
// print out to the terminal the internal configuration of the PLC
String plcName = "PLogic:E:36";
String propPosition = "PointerPosition";
String propCellType = "EditCellCellType";
String propCellConfig = "EditCellConfig";
String propCellInput1 = "EditCellInput1";
String propCellInput2 = "EditCellInput2";
String propCellInput3 = "EditCellInput3";
String propCellInput4 = "EditCellInput4";

for (int cellNr=1; cellNr<=16; cellNr++) {
   mmc.setProperty(plcName, propPosition, cellNr);
   print("Cell " + cellNr + ":");
   print("  type: " + mmc.getProperty(plcName, propCellType));
   print("  config: " + mmc.getProperty(plcName, propCellConfig));
   print("  inputs: " + mmc.getProperty(plcName, propCellInput1) +
", " +
   mmc.getProperty(plcName, propCellInput2) + ", " +
   mmc.getProperty(plcName, propCellInput3) +  ", " +
   mmc.getProperty(plcName, propCellInput4) );
}
```

```
for (int ioNr=1; ioNr<=16; ioNr++) {
   mmc.setProperty(plcName, propPosition, ioNr + 32);
   print("I/O " + ioNr + ":");
   print("  type: " + mmc.getProperty(plcName, propCellType));
   print("  addr: " + mmc.getProperty(plcName, propCellConfig));
}
```

Hardware Specifications

Input/Output Impedance

As described above the BNC impedance values for Rev B boards are as follows: input: 27 kOhm, open-drain output: 43 ohm in series, push-pull output: approximately 50 ohm The backplane I/Os are implemented using pull-up logic with 10 kOhm resistors on the board. The front panel I/Os are protected with 43 ohm in series and a 27 kOhm to ground (100 ohm and 10 kOhm in Rev A, except for a very few early cards with the 10 kOhm resistor to 5V instead).

Logic Thresholds

Push-pull output guaranteed to be < 0.44 V logic low (typically < 0.1 V) and > 4.25 V logic high (typically > 4.9 V) with 5.0 V output level selected with a jumper as per default.

Guaranteed input thresholds are < 1.0 V for logic low and > 2.3 V for logic high.

diSPIM backplane

The standard TG-1000 backplane is configured as follows for diSPIM:

PLC Address	**Signal6**	**diSPIM use**
41	TTL0	Camera Path A
42	TTL1	Laser 0 (laser trigger)
43	TTL2	Camera Path B
44	TTL3	Laser 1 (laser side)
45	TTL4	Piezo Path A
46	TTL5	Trigger Input
47	TTL6	Piezo Path B
48	TTL7	Reserved

Laser 0 and Laser 1 outputs behave according to a firmware setting (`LED Z`). This setting is usually changed by the user to match the laser control configuration they are using. The outputs can be set as:

- individual laser on/off for the two paths
- single laser trigger with side switch (Laser 0 is laser on/off for both sides and Laser 1 is high when Path B is active) (default)
- side indication only (Laser 0 high when Path A is active, Laser 1 high when Path B is active)

Implementation Notes

Originally it was hoped to run the evaluation cycle at a very fast update rate, e.g. at 100kHz, so that any propagation time would be negligible. However, that proved unfeasible. Instead we provide a mechanism to slave the outputs to another card's clock.

Every indication is that sustained 4kHz operation is possible for all possible logic cell assignments (i.e. <0.25 ms per cycle), but a rigorous evaluation has not been performed. With all 4-input LUTs it took 0.12ms to do the evaluation cycle, and with all constants it took 0.07ms. With 32 programmable cells, all 4-LUTs, it took 0.20ms, indicating that about 0.04ms is overhead and about 5us per cell. Same experiment with constants, the time is about 0.03ms overhead and 5us per cell. Dec 2018 in experiment with diSPIM shutter use of PLogic card using Micro-Manager it was found to require ~0.13ms for evaluation.

Outputs can be used immediately when evaluating subsequent cells. This makes it possible to implement some functionality with less delay (e.g.the ripple counter would not have any effective “ripple” time, just a single cycle, as long as the higher-order bits are placed in cells evaluated after the lower-order bits). However, note that the logic's behavior can depend on cell ordering. In each evaluation cycle, cell 1 is evaluated first, then cell 2, and so on.

Additional cell types can easily be added.

Additional presets can easily be added.

Additional programmable cells can also be added but proceed with caution to make sure an evaluation cycle can be completed in the allotted time. As many as 32 cells may be doable with 0.25ms clock rate, and we have pre-allocated addresses for 32 cells. The default build includes 16 cells but a build with 24 cells is available as of March 2019; contact ASI if you need it.

Serial Command Cheatsheet

The follow table assumes that the axis letter for the PLogic card is E. Axis specific commands have the axis character highlighted in bold.

The second section in the table has commands that operate on the logic cell or physical I/O at the current pointer position.

Note that the `CCA Y` and `CCA Z` commands change behavior based on the current pointer position.

The first two entries of the table are not card addressed, so `M E=1` moves the pointer position to cell 1, and `W E` queries the pointer position.

Tiger card: prefix the card address to the command. For example, if the card address is 6, use `6CCA Z=64` to set the current cell configuration to 64.

PLogic Serial Commands				
Property	**Set**	**Get**	**Notes**	**Requirements**
Clear All Cell States	! **E**	-	Not card addressed	Check Table for “Has State?”
Pointer Position	M **E**=#	W **E**	Not card addressed	
Trigger Source	PM **E**=#	PM **E**?	Clock source	
Card Preset	CCA X=#	-	Set card preset	
Front Panel Output State	-	RA X?	Front panel I/O	
Backplane Output State	-	RA Y?	TTL backplane	
Lower Cells Output State	-	RA Z?	Cells 1-16	
Upper Cells Output State	-	RA F?	Cells 17-32	
Pointer Position - Logic Cell or Physical I/O				
I/O Type	CCA Y=#	CCA Y?	Input, push-pull, etc	Pointer Position > Number of Cells
Source Address	CCA Z=#	CCA Z?	Input source address	Pointer Position > Number of Cells
Cell Type	CCA Y=#	CCA Y?	Delay, one-shot, etc	Pointer Position ≤ Number of Cells
Cell Configuration	CCA Z=#	CCA Z?	Configuration value	Pointer Position ≤ Number of Cells
Cell State	CCA F=#	CCA F?	Internal cell state	Check Table for “Has State?”
Cell Input 1	CCB X=#	CCB X?	Edit input value 1	
Cell Input 2	CCB Y=#	CCB Y?	Edit input value 2	
Cell Input 3	CCB Z=#	CCB Z?	Edit input value 3	
Cell Input 4	CCB F=#	CCB F?	Edit input value 4	

Logic Cell: Pointer Position ≤ Number of Cells; Physical I/O: Pointer Position > Number of Cells

plc, manual, tiger

1) The number of logic cells is easily extendable, but then longer evaluation times are required. It may be beneficial to let the user say how many logic cells are used.

2) It is easily extendable to 6 inputs, but 4 is enough for logic cells we have now.

3) not strictly simultaneous but within 0.3us of the trigger (had 0.3 with internal triggering, measured a bit less than 1 us with with external)

4) again, not strictly synchronously but within 0.5us of the trigger

5) An additional 8 backplane connectors are reserved but not used at present.

6) Really the OR gate's address + 128 will be assigned as the flip-flop's input address, but it is equivalent in function.

7) to have the output high for only a single evaluation cycle, connect the delay cell's output to its own reset input

8) Prior to v3.05 firmware the BNCs were also inputs by default.

9) Technically speaking the push-pull buffer is not rated for supply as low as 3.3V but we have never experienced any problems operating it this way.

10) Strictly speaking the series resistor mentioned later also is part of the input impedance.

11) For Rev A boards, the micro-controller itself was set to push-pull output and would drive the BNC

connector through a resistor. For Rev B and later boards, an additional buffer is used in push-pull mode that is capable of driving a 50ohm load.

[12]

Prior to v3.09 BNC3 was set to the equivalent of preset 24 as well

[13]

Note that there will be 0.25ms jitter because the internal stage sync signal is asychronous but the PLC is (almost always) sampling/outputing its signals on a 4kHz clock. The corresponding positional jitter depends on the speed, e.g. at 40 um/s speed the positional jitter will be 0.25ms * 40 um/s = 10nm which is negligible.